Heterogeneous Secure Multi-Party Computation

Mentari Djatmiko[12], Mathieu Cunche[1], Roksana Boreli[12], and Aruna Seneviratne[12]

[1] NICTA, 13 Garden Street, Eveleigh, NSW, 2015, Australia
[2] University of New South Wales, Sydney, NSW, 2052 Australia
{mentari.djatmiko,mathieu.cunche
roksana.boreli,aruna.seneviratne}@nicta.com.au

Abstract. The increased processing power and storage capacity of in-home and mobile computing devices has motivated their inclusion in distributed and cloud computing systems. The resulting diverse environment creates a strong requirement for secure computations, which can be realised by Secure Multi-Party Computation (MPC). However, MPC most commonly assumes that parties performing the secure computation have the same characteristics and evenly distributes the computation load. In a heterogeneous environment, MPC using the same approach would result in poor performance. In this paper, we propose a mechanism for MPC share distribution in such an environment and present an analysis of the gain in robustness and the corresponding computational and communication complexity. Our results show that the uneven share distribution is a worthwhile approach in diverse computing systems.

Keywords: secure multi-party computation, distributed computing, unequal shares, heterogeneous platforms

1 Introduction

The increasing popularity of distributed computing in latter years has resulted in a number of developments which use diverse distributed computing platforms [12], [3]. Cloud computing adds virtualisation and additional management functionalities to distributed computing and cloud services have become popular and are readily available from a number of providers (e.g. Amazon, Google). Community clouds may be provided by multiple organisations, or private individuals. Finally, recent research proposals [14] include cloud systems which bring together in-home, mobile and data centre based computing resources, thereby adding the continuously growing computational capacity from various personal and mobile computing devices to the overall pool of computing resources.

The distributed computing systems may be used for a wide range of applications and services, including distribution and editing of media content, content-rich social networks, gaming, data storage, home security and others. In the majority of such applications, the data may be commercially sensitive or of personal nature e.g. family videos or images captured by a home security system.

This creates a strong requirement for processing of data in a secure and private way by the distributed computing system. The argument for security and privacy is further strengthened by the increasingly high level of diversity of the distributed systems, which includes different computing devices, owners of such devices and environments in which they are operated.

There have been a number of proposals that address the mechanism to process sensitive or private data on distributed computing systems. The most widely used technique to address this is obfuscation of data by encrypting/encoding the values and performing a set of operations on the encrypted data [6, 18, 7]. Of these, Secure Multi-Party Computation (MPC) [7] has the benefits of provable security under specific adversary models where the security is comparable to an ideal scenario where data is processed by a trusted third party. Furthermore, in MPC, processing parties can compute over the the encoded data without having a knowledge of the original data.

MPC most commonly assumes an equal distribution of encoded data (i.e. shares) between the parties performing the secure processing [5]. This is a valid assumption when these parties have similar capabilities and a similar level of trust related to performing the required computations and the level of security provided against malicious attacks. However, in the distributed computing platforms which consists of heterogeneous entities, the capabilities, trust and security of individual devices may be highly variable. Therefore, having equal sharing of the MPC data and processing between such diverse participants should logically result in a substandard performance.

In this paper we propose a method of uneven sharing of data and processing in the MPC system, which further extends the hierarchical MPC proposals [4]. Particularly, we present the concept of uneven distribution of encoded data between participants performing secure computation and make the following contributions. We propose a generalised MPC protocol which enables the use of unequal share distribution among the computing devices, according to a given set of criteria. We demonstrate the improved MPC computation integrity and data privacy failure tolerance, as compared to the case when all participants have a single share. We show that, with an appropriate choice of parameters, the unequal distribution of shares does not adversely affect the performance of MPC through increased complexity and communication overhead.

The paper is organised as follows. The overview of MPC and the related works are presented in Sect. 2. Section 3 describes MPC protocol for heterogeneous environment. In Sect. 4 we discussed the performance of the proposed protocol, including methods to allocate shares, while in Sect. 5 we evaluate the overhead induced by unequal share distribution. We conclude in Sect. 6.

2 MPC Overview and Related Works

MPC enables multiple entities to jointly compute a mathematical function which takes inputs from a number of contributing entities, and provides a formal guarantee of the privacy of the input data and the correctness of the computed result

[7]. The two-participant version of MPC was originally proposed by Yao [19] as the millionaire's problem. Subsequent research efforts focus on improving the security (e.g. [10]) and implementing MPC (e.g. SEPIA in [5]).

MPC utilises other cryptographic schemes in its protocol. One important scheme is secret sharing, which protects the privacy of the input data. The most generally applicable secret sharing scheme, which we focus on in this paper, is Shamir's scheme [16] which uses polynomial interpolation. We note that other secret sharing methods may also be applicable [8]. The concept of distributing unequal number of shares to participants based on the participants characteristics (e.g., level of authority) is briefly discussed by Shamir in [16] which is referred to as hierarchical scheme. However, this concept has not yet been utilised in MPC. In this paper, we provide an approach which apply this concept in MPC and evaluate the performance by comparing our approach to the baseline mechanism where each participant receives a single share.

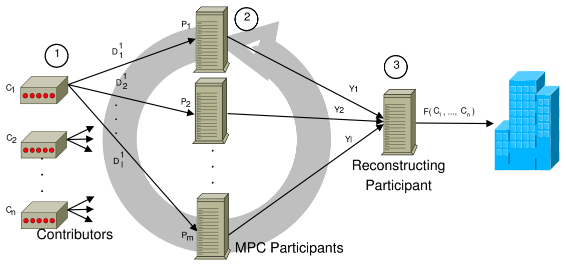


Fig. 1. Illustration of SMPC

An example MPC system using secret sharing is shown in Fig. 1. The MPC protocol consists of the three following stages:

1. **Input sharing:** a number of contributors provide input data for secure computation. The input data are shared with the participants using secret sharing scheme as discussed before. As shown in Fig. 1, each secure computing entity may be allocated one or more shares.
2. **Arithmetic operations:** the participants compute a mathematical function using the inputs they have received. **Addition of two values** only requires addition of two shares since the two are equivalent. **Multiplication of two values** is a more complex operation, and re-sharing and reconstruction steps are required after the local multiplication of input data shares.
3. **Result reconstruction:** one participant or an independent entity is in charge of reconstructing the result. This is done by by computing a linear combination of the participants outputs, where the linear combination is obtained from a polynomial interpolation algorithm, e.g., Lagrange interpolation and Neville's algorithm.

MPC has a well defined notion of adversaries and their effect on the security and privacy of the computations [7]. The participants which are involved in the attack on the MPC system (adversaries) are considered to be *corrupt*, while the rest of the participants are considered to be *honest*. Attacks can be classified into passive attacks, where the adversaries attempt to breach the privacy of the data while still conforming to the protocol, and active attacks, where the adversaries do not conform to the protocol thus compromising the result integrity. As discussed in [2], there are two possible attack models to be considered for corrupt participants: collusion attack, in which corrupt participants collude to reveal the result of computation and thus negate the privacy of MPC; integrity attack where corrupt participants can also alter the intermediate computation result. For Shamir's secret sharing, privacy can be preserved if the number of corrupted participants are less than half of the total number of participants, while the result integrity can be guaranteed if the number of corrupted participants is less than one third of this number [2].

3 MPC in a Heterogeneous Environment

We consider an application which has sensitive data and is processed on a heterogeneous distributed computing platform, with a number of diverse devices and players which operate the devices. Various aspects of heterogeneity include different security levels within various participating companies or personal (including in-home) computing devices, trust levels on conforming to the agreed collaboration protocol, and reliability variations of the included devices.

Having equal sharing of the data and processing between such diverse participants, as is commonly done in MPC [5], is likely to result in substandard MPC performance. This motivates our interest in exploring uneven sharing of data and processing in a heterogeneous MPC system. The most obvious consequence of unequal share distribution is that a participant may receive more than a single share. Therefore, together with the potential performance gain, we also need to explore the corresponding processing complexity and communication overhead.

Table 1. Description of variables

k	number of shares required or the threshold to recover the secret
n	total number of contributors (i.e. input data provider)
m	total number of MPC participants
l	total number of shares in the system
r	resolution, describes the relationship between m and l
l_i	number of shares received by participant i from each contributor
c	number of corrupt participants
P_i	participant i
D^j	input from contributor j
D_i^j	share from contributor j's input with index i

Table 1 lists the variables used in the paper. An MPC system consists of n contributors (that provide the input data D^i) and m participants (that perform MPC protocol). An entity can be a contributor, a participants or both. Each of the n contributors generates l shares and distributed the shares over

m participants, where each participant receives l_i shares from each contributor. To simplify the analysis, we define resolution, r, the average units of shares per participant which describes the relationship between m and l according to $l = r \times m$. In equal share distribution, $l = m$ and $l_i = 1\ \forall i$ while in unequal share distribution, $l > m$ and $l_i \geq 1\ \forall i \in [1..k-1]$. We also introduce the notion of share index, which refers to the point where the polynomial is evaluated to obtain the share. It is important that the contributors have agreed on l_i that is allocated to each P_i before the protocol begins, to ensure the mathematical function to be properly executed.

3.1 MPC Protocol for Unequal Share Distribution

For unequal share distribution, MPC protocol, described in Sect. 2, needs to be modified to accommodate allocating of multiple shares to selected participants and processing of multiple shares. Note that, while we only consider addition and multiplication of two values in this paper, our method is also applicable to other arithmetic operations.

For **input sharing**, unequal share distribution requires contributors to generate a larger number of shares, since $l > m$. Similarly, **addition of two values** requires each participant to repeat the addition operation l_i times.

Algorithm 1: Multiplication of two values for P_i

```
   Data: Two vectors of shares from two contributors: $[D^\alpha]$ and $[D^\beta]$
   Result: Multiplication result vector: $[D^{\alpha\times\beta}]$
 1 for $i \in l_i$ do
 2     $d_i^{\alpha\times\beta} = D_i^\alpha \times D_i^\beta$
 3     Generates $l$ shares for $d_i^{\alpha\times\beta} = \{d_{i,1}^{\alpha\times\beta}, ..., d_{i,l}^{\alpha\times\beta}\}$
 4     for $j = 1$ to $l$ (except for $j = l_{first}, ..., l_{last}$) do
 5         Sends $d_{i,j}^{\alpha\times\beta}$ to the corresponding participant
 6     end
 7 end
 8 Computes recombination vector $[r] = [r_1, ..., r_l]$ using polynomial interpolation
 9 for $i \in l_i$ do
10     Recovers the final multiplication results: $D_i^{\alpha\times\beta} = \sum_{j=1}^{l} r_j \times d_{j,i}^{\alpha\times\beta}$
11 end
```

We present details of the **multiplication of two input values** for unequal share distribution MPC in Algorithm 1. First, each participant needs to locally multiply two shares with the same share index for each of the l_i shares. Each participant then re-share each local multiplication result to all other participants. In the final step, each participant needs to reconstruct the result. To do this, the participant computes the recombination vector, $[r]$ using interpolation to solve the polynomial of size l. The final result for each l_i is then recovered by summing up $d_{1,i}^{\alpha\times\beta}, ..., d_{l,i}^{\alpha\times\beta}$ with $[r]$ as weights.

For **result reconstruction** in unequal distribution MPC, each participant must send l_i outputs to the reconstructing entity. The recombination vector computation and result recovery are equivalent to the operations in MPC with equal share distribution.

4 Share Allocation and its Impact

Unequal share distribution in MPC aims to minimise the probability of data privacy breach and the integrity failure of the result. Intuitively, participants which are allocated a larger number of shares should be less likely to cause the privacy and integrity failures. The adversary models presented in Sect. 2 relate to the numbers of honest and corrupt participants. In reality, the participants may have a probability of being honest or corrupt, which can be related to the combination of trustworthiness and computing failure probability and integrity. By trustworthiness, we consider the likelihood that the participant will conform to the MPC protocol, i.e. will not collude to breach the privacy or intentionally provide an incorrect individual computation result. Computing failure probability quantifies the likelihood that a participant will (unintentionally) alter the result integrity based on the reliability of it's computing platform. Trustworthiness and computing failure probability can be derived from an associated trust mechanism using e.g. a reputation approach [15], in which the participants provide feedback on the integrity of transactions and from participant's hardware and operating system specification.

4.1 Probability of Integrity and Privacy Failure

MPC result integrity failure occurs when the number of corrupt participants exceeds the threshold number $c \geq \frac{m}{3}$ (see Sect. 2). More generally, MPC result integrity is guaranteed when the total number of shares belonging to the corrupt participants is less than k, where $k = \frac{l}{3}$. Similarly, privacy can be formally guaranteed when the threshold related to the number of shares controlled by corrupt participants is less than $k = \frac{l}{2}$. In the following paragraphs we address the probability of generic MPC failure, which relates to both integrity and privacy.

Let us assume that each participant P_i has an associated probability of being corrupt p_{ci}. We first need to consider all possible combinations of corrupt participants, obtained by computing all possible subsets of the participants. For each combination, s, the joint probability that all participants in the subset s are corrupt, while the remainder of the participants are honest (assuming the participants are independent), $p_c(s) = \prod_{i \in s} p_{ci}$, can be computed. Once the share distribution is known, we can determine which of the corrupt cases will result in integrity or privacy failure, i.e., when $l_s \geq k$ where $l_s = \sum l_i \in s$. The failure probability can be derived as the sum of the probabilities of these corrupt cases.

Given $f(s)$ as a function determining whether s results in MPC failure, the probability of failure p_{fail} can be calculated as:

$$p_{fail} = \sum_{\forall s} p_c(s) \times f(s). \qquad (1)$$

where $f(s) = 1$ if $l_s \geq$ k, otherwise $f(s) = 0$. p_{fail} is independent of the share distribution or the algorithm used to generate the distribution.

4.2 Share Distribution Algorithms

The most direct method to find a share distribution that would minimise the failure probability is **exhaustive search**. While it guarantees the optimal result, it is highly inefficient as the complexity[3] of the search is m^l. Therefore we discuss two other share distribution algorithms which are more practical while not guaranteeing the globally optimum result.

Heuristic method linearly estimates l_i based on p_{ci}. The method calculates $p_{hi} = 1 - p_{ci}\ \forall P_i$ and then normalises each p_{hi} by dividing it with the smallest p_{hi}. The ideal number of shares can be obtained assuming that $l_i = 1$ for P_i with the smallest p_{hi} and for a given l, l_i can be obtained by scaling the ideal number of shares.

Genetic algorithm (GA) [1] is a stochastic optimisation technique inspired by natural evolution which provides a trade-off between the optimum result and complexity. GA iteratively searches for the optimum solution in an evolving population of possible share distribution. The initial population is comprised of randomly generated share distributions. The population is evolved by combining and mutating the share distributions as well as retaining a few share distributions which have low p_{fail}. The algorithm is terminated when either the optimal solution of several consecutive generations of the populations are identical or a fixed number of iterations have been reached.

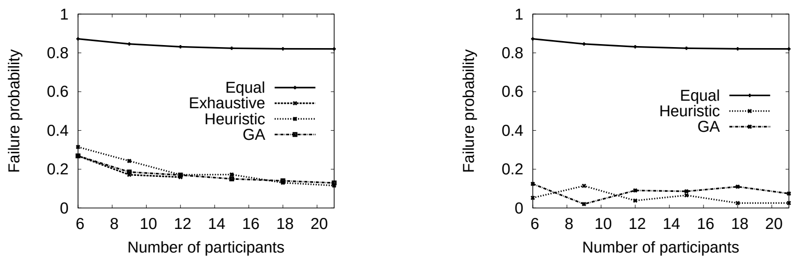
4.3 Performance Evaluation

The gain of unequal share distribution MPC is evaluated by simulations based on original implementation in C++. We have not performed an analytical evaluation since finding optimal share distribution for unequal MPC is essentially a combinatorial optimisation problem which does not have an analytical solution.

Simulation, Results and Discussion: We consider a scenario where the participants can be divided into two classes: resilient participants which have low p_{ci} (close to fully honest) and vulnerable participants which have high p_{ci} (close to corrupt). We note that this represents the upper bound on the potential performance gains compared to the unequal share distribution, and that considering participants with closer corrupt probabilities may result in lower performance improvement. We present only the results for MPC integrity failure that is for $k = \frac{l}{3}$, as the improvement for privacy failure only relates to a threshold change.

In the simulations, we vary m, r and the share distribution algorithm. m is increased from 6 to 21, higher m is not simulated due to high complexity. We use $r = 2, 5, 10$ for unequal share distribution MPC, where $r = 2$ represents the smallest possible resolution and $r = 5, 10$ represent higher resolutions. Note that r can be any positive integer. The performance of equal share distribution is considered as a baseline, which we compare to the performance of the exhaustive search, heuristic method and GA share distribution algorithms. For resilient participants $p_{ci} = 0.1$ and for the remaining participants $p_{ci} = 0.9$.

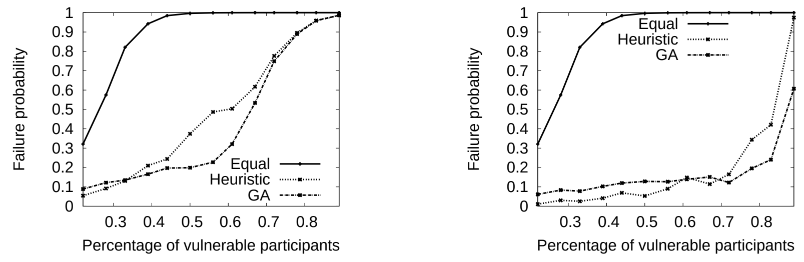
[3] The complexity depends on the representation of share distribution, which in this case is by the number of shares allocated to each participant.



(a) Resolution 2 (b) Resolution 5

Fig. 2. Failure probability of MPC with different share distribution algorithms for resolution 2 and 5 in the scenario where a third of the participants are vulnerable

Fig. 2(a) and 2(b) present the failure probability results for $r = 2$ and $r = 5$. We can see that p_{fail} for unequal share distribution MPC is significantly lower regardless of the share distribution algorithm used. Due to high complexity, exhaustive search has been omitted for the subsequent evaluation, but it is apparent from Fig. 2(a) that the performance of GA (and the heuristic method for $m = 12$) closely follows exhaustive search. Fig. 2(b) shows the result for $r = 5$. In this case, heuristic method performs slightly better than GA. The graph for $r = 10$ is not included as the result is similar to $r = 5$ in this particular scenario.



(a) Resolution 2 (b) Resolution 5

Fig. 3. Failure probability of MPC when the total number of participants is 18 and the number of vulnerable participants is varied between 4 and 16

The performance improvement of unequal share distribution is demonstrated in Fig. 3(a) and Fig. 3(b). We can observe that the failure probability of equal distribution MPC rapidly increases as the percentage of corrupt participants increases. Complete failure (i.e. $p_{fail} = 1$) for equal share distribution occurs when at least 45% of the participants are vulnerable. On the other hand, unequal share distribution MPC suffers complete failure when there are close to 90% vulnerable participants in the system. The resolution only affects the rate of change of p_{fail}, where the rate of change is higher when $r = 2$ compared to when $r = 5$. As can be logically expected, this demonstrates the improved resilience of unequal share distribution compared to equal distribution MPC to the number of low trustworthy participants.

Table 2. Group computational complexity

Operations	Computational	
	Equal	Unequal
Input sharing	$n.m(m-1)$	$n.r.m(r.m-1)$
Addition of two input values	m	$r.m$
Multiplication of two input values	$2m^3+m^2$	$(r^2+r^3)m^3+r^2.m^2$
Result reconstruction	$(\frac{m^2}{9}+\frac{2m}{3}-1)$	$(\frac{(r.m)^2}{9}+\frac{2r.m}{3}-1)$

Table 3. Group communication complexity

Operations	Communication	
	Equal	Unequal
Input sharing	$n.m$	$n.r.m$
Multiplication of two input values	$m(m-1)$	$r^2.m(m-1)$
Result reconstruction	m	$r.m$

While we evaluate a specific scenario, this result can be applied to a more generic case where each participant has different p_{ci}.

5 Heterogeneous MPC Overhead

In this section we analyse the computational and communication complexity of both the equal and the unequal share distribution in MPC. The analysis and the numerical evaluation for a selected range of parameters is presented for the aggregate of all MPC participants. We then discuss the tradeoff between the performance improvements and the introduced overhead and recommend a range of parameters which provide a good balance between the two.

5.1 Complexity Analysis

The computational and communication complexities are summarised in Tables 2 and 3. As variables such as k and l can be reduced to m, n and r, we present the complexities with respect to the latter set of variables. We define C_{comp} and C_{comm} respectively as group computational and communication complexity. Note that $k = \frac{l}{3}$ and $\sum_{i=1}^{m} l_i = l$. For equal share distribution $l = m$ and for the unequal distribution $l = r.m$. All operations are assumed to have the same computational complexity. Finally, for share generation using polynomial interpolation is $C_{comp} = 3k - 1$.

Input sharing: n contributors generate and distribute l shares. In total, there are nl shares generated and distributed in the system. The computational complexity per contributor can be calculated as $l(3k-1)$ and for the group is $n.l(3k-1)$. For equal distribution MPC, $C_{comp} = n.m(m-1)$ and for unequal distribution MPC $C_{comp} = n.r.m(r.m-1)$. The communication complexity for each contributor is proportional to l_i (see Table 1) and for the group is equal to the total number of shares in the system, l. Therefore, $C_{comm} = m$ for equal distribution MPC and $C_{comm} = r.m$ for unequal distribution MPC.

Addition of two input values: the number of computations that each participant performs is proportional to l_i and the total number of computations for the group is equal to l. For equal share distribution MPC, $C_{comp} = m$, and for unequal share distribution MPC, $C_{comp} = l = r.m$. Note that this operation does not require any communications.

Multiplication of two input values: note that the complexity of computing the recombination vector using polynomial interpolation is quadratically proportional to the degree of the polynomial [13]. For each participant, the number of computations required for equal and unequal distribution MPC depends on l_i. For equal distribution MPC, each participant needs to compute local multiplication (one computation), re-sharing of the local multiplication result ($l(3k-1)$ computations), computing recombination vector of degree-l polynomial (l^2 computations) and to recover the result using weighted sum ($2l-1$ computations). In total, each participant conducts $l^2 + 3l.k + l$ operations and the corresponding complexity for a group of m participant is $C_{comp} = 2m^3 + m^2$. For unequal share MPC, each participant needs to repeat local multiplication, re-sharing and result recovery l_i times. In total, each participant calculates $l^2 + 3l_i.l.k + l_i.l$ and therefore for the group $C_{comp} = (r^2 + r^3)m^3 + r^2.m^2$.

Communication occurs during the re-sharing step. The communication overhead for individual participant in equal MPC is $l-1$ and hence $C_{comm} = m(m-1)$. For unequal MPC, each participant distributes $l_i(l-l_i)$ messages, resulting in $C_{comm} = \sum_{i=1}^{m} l_i(l-l_i)$ which depends on the share distribution. To simplify the analysis in the next Subsection, we consider the case where, on the average $l_i = r$ which gives $C_{comm} = \sum_{i=1}^{m} r(l-r) = r^2.m(m-1)$.

Result reconstruction: the share distribution does not affect the computational complexity as the result is reconstructed by an entity (which is either one of the participants or an independent entity). C_{comp} is the sum of the complexity of generating the recombination vector (k^2) and computing the weighted sum ($2k-1$). Therefore the group complexity is given by $k^2 + 2k - 1$. For equal distribution MPC, $C_{comp} = \frac{m^2}{9} + \frac{2m}{3} - 1$ and for unequal distribution MPC, $C_{comp} = \frac{(rm)^2}{9} + \frac{2rm}{3} - 1$.

Each participant needs to send the outputs to the reconstructing entity. The communication overhead per participant in result reconstruction is equal to l_i. Furthermore, the communication overhead for the group is equal to the total number of shares in the system. Hence, for equal distribution MPC $C_{comm} = m$ and for unequal distribution MPC $C_{comm} = l = r.m$.

5.2 Numerical Evaluation

We numerically evaluate the group computation and communication overhead, defined as the ratio of the complexity of the unequal distribution MPC to the equal distribution MPC, i.e. the increase in operations or messages required. m is varied between 6 and 120, and for unequal MPC $r = 2, 5, 10$.

Fig. 4(a) to 4(d) shows the overhead of unequal distribution MPC as m increases. In all the graphs, the ratio converges to a value, which can be obtained by dividing the complexity of unequal distribution MPC to equal distribution MPC. For input sharing, the group overhead ratio is represented by Fig. 4(a), where the ratio approaches r^2 as m increases. Fig. 4(b) shows the overhead ratio for addition of two values. We can observe that the ratio is equal to r, which is expected from the results in Table 2. The group overhead for multiplication of two values is presented in Fig. 4(c). The trend shows that the ratio approaches $\frac{r^3+r^2}{2}$ for large m. Finally, Fig. 4(d) presents the overhead ratio for result reconstruction. Similar to input sharing, the ratio approaches r^2 when m increases.

The communication complexity in Table 3 shows that the ratio for input sharing and result reconstruction is proportional to r while for multiplication of two values the overhead ration is r^2.

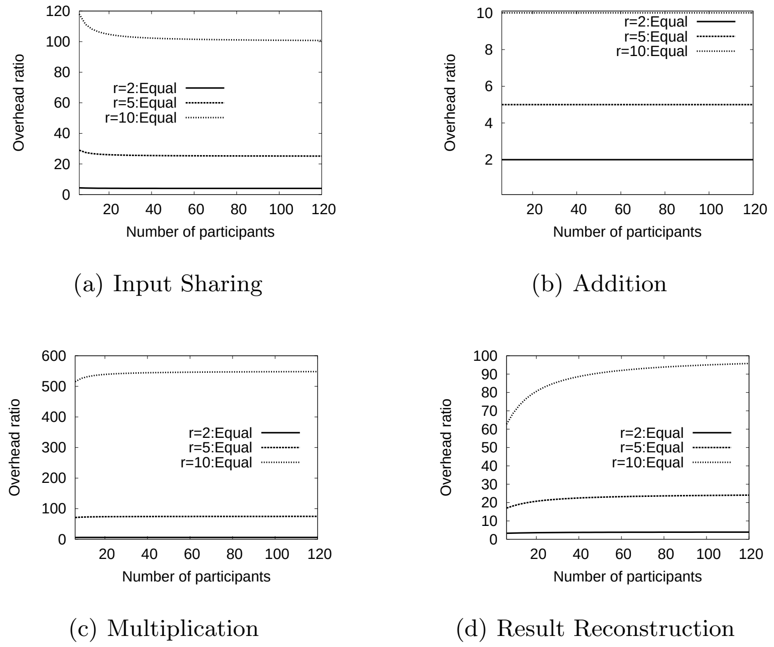


Fig. 4. Group computational overhead in terms of the ratio of the computational complexity of the unequal distribution MPC to the equal distribution MPC

5.3 Discussion

We estimate the computation time of MPC operations on mobile devices to evaluate the practicality of unequal share distribution. The multiplication of two values is selected as it has the highest computational complexity among MPC operations. The estimated computation time presented in Table 4 is computed by using the expressions in Table 2 and the computation power of the mobile device obtained using Linpack [11] benchmark (e.g., Android Nexus S device has the computation power of 17 MFLOPS). While the MPC operations may not always be floating point, this provides the upper bound on the expected values as the floating point operations require more time to compute than integer operations.

Table 4. Estimated computation time for the multiplication of two values (ms)

m \ r	1	2	5	10
6	0.0046	0.027	0.33	2.36
12	0.018	0.10	1.29	9.39
30	0.11	0.64	8.0	58.41
60	0.43	10.19	127.24	932.47

The system is still considered to be practical if the computation time is below or close to the Internet latency. Assuming that MPC participants are located worldwide, the average latency is estimated to be 100 ms [17]. As can be seen from Table 4, the computation time for small m is still well below the average latency even for high resolution. The computation time becomes very high when both m and r are high. The increase in communication load is considered less critical than the complexity, as the size of MPC protocol messages is small [9].

Although the full impact of unequal distribution MPC will depend on the protocol implementation and the computing platforms, we have clearly demonstrated the improved performance of unequal distribution compared to that of equal distribution on heterogeneous platforms. While the computation and communication overheads increase with r, it is not always the case for the robustness (see Sect. 4). Hence, it is important to carefully select the value of r.

6 Conclusion

We explore unequal share distribution in MPC used on heterogeneous platforms and demonstrate that it can significantly improve the system's robustness compared to the standard case where all participants have the same number of shares. However, such an improvement comes with the increase in complexity and overhead. Therefore, we also consider the trade-off between the two in the choice of parameter values. As a future work, we plan to investigate the methodology to reduce the complexity and overhead of unequal share distribution in MPC.

References

1. Alba, E., Cotta, C.: Evolutionary algorithms. In: Handbook of Bioinspired Algorithms and Applications, chap. 2, pp. 3–19. Chapman & Hall (2006)
2. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: Proceedings of the 20th annual ACM Symposium on Theory of computing. pp. 1–10. ACM, New York, NY, USA (1988)

3. BOINC author: Boinc, http://boinc.berkeley.edu/wiki/System _requirements
4. Brickell, E.: Some Ideal Secret Sharing Schemes. In: Advances in Cryptology, LNCS, vol. 434, chap. 45, pp. 468–475. Springer Berlin / Heidelberg, Berlin, Heidelberg (May 1990)
5. Burkhart, M., Strasser, M., Many, D., Dimitropoulos, X.: SEPIA: privacy-preserving aggregation of multi-domain network events and statistics. In: Proceedings of the 19th USENIX conference on Security. p. 15. USENIX Association, Berkeley, CA, USA (2010)
6. Cao, N., Yang, Z., Wang, C., Ren, K., Lou, W.: Privacy-preserving query over encrypted graph-structured data in cloud computing. In: ICDCS, 2011 31st International Conference on. pp. 393–402 (june 2011)
7. Cramer, R., Damgaard, I., Nielsen, J.B.: Multiparty Computation, an Introduction (May 2008)
8. Cramer, R., Damgard, I., Maurer, U.: General secure multi-party computation from any linear secret-sharing scheme. In: Advances in Cryptology, LNCS, vol. 1807, pp. 316–334. Springer Berlin / Heidelberg (2000)
9. Damgaard, I., Geisler, M., Krøigaard, M., Nielsen, J.B.: Asynchronous Multiparty Computation: Theory and Implementation. In: Proceedings of the 12th International Conference on Practice and Theory in Public Key Cryptography: PKC '09. pp. 160–179. Springer-Verlag, Berlin, Heidelberg (2009)
10. Damgrd, I., Desmedt, Y., Fitzi, M., Nielsen, J.B.: Secure protocols with asymmetric trust. In: Advances in Cryptology - ASIACRYPT 2007. pp. 357–375 (2007)
11. Dongara, J.: Linpack for android, http://www.netlib.org/benchmark/linpackjava/
12. Estrin, D.: Participatory sensing: applications and architecture [internet predictions]. Internet Computing, IEEE 14(1), 12–42 (jan-feb 2010)
13. Goldman, R.: Pyramid Algorithms: A Dynamic Programming Approach to Curves and Surfaces for Geometric Modeling, chap. 2. Lagrange Interpolation and Neville's Algorithm. Morgan Kauffman (2003)
14. Kannan, S., Gavrilovska, A., Schwan, K.: Cloud4home – enhancing data services with @home clouds. In: Distributed Computing Systems (ICDCS), 2011 31st International Conference on. pp. 539 –548 (june 2011)
15. Michiardi, P., Molva, R.: Core: a collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks. In: Conference on Communications and Multimedia Security. p. 121. Kluwer, BV (2002)
16. Shamir, A.: How to share a secret. Commun. ACM 22(11), 612–613 (Nov 1979)
17. Verizon: Ip latency statistics (2011), `http://www.verizonbusiness.com/about/network/latency/`
18. Wang, C., Ren, K., Wang, J., Urs, K.: Harnessing the cloud for securely solving large-scale systems of linear equations. In: ICDCS, 2011 31st International Conference on. pp. 549–558 (june 2011)
19. Yao, A.C.: Protocols for secure computations. Foundations of Computer Science, Annual IEEE Symposium on pp. 160–164 (1982)